Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/GB05/000964

International filing date: 14 March 2005 (14.03.2005)

Document type: Certified copy of priority document

Document details: Country/Office: GB
Number: 0417762.2
Filing date: 10 August 2004 (10.08.2004)

Date of receipt at the International Bureau: 02 May 2005 (02.05.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

PCT/GB 2005 / 0 0 0 9 6 4

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed William Morell

Dated 12 April 2005

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
CF
10 AUG 2004
RECEIVED BY FAX

The Patent Office

10AUG04 E917795-1 D10153
P01/7700 0.00-0417762.2 NONE

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office
Cardiff Road
Newport
South Wales
NP10 8QQ

1. Your reference — P188-GB

2. Patent application number *(The Patent Office will fill in this part)* — 0417762.2

3. Full name, address and postcode of the or of each applicant *(underline all surnames)* — 1... Limited
St John's Innovation Centre
Cowley Road
Cambridge CB4 0WS

Patents ADP number *(if you know it)* — 8113870001

If the applicant is a corporate body, give the country/state of its incorporation — England

4. Title of the invention

AUTOFOCUS METHODS

5. Name of your agent *(if you have one)* — Ursula Lenel

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)* — 1... Limited
St John's Innovation Centre
Cowley Road
Cambridge CB4 0WS

Patents ADP number *(if you know it)*

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*
   a) any applicant named in part 3 is not an inventor, or
   b) there is an inventor who is not named as an applicant, or
   c) any named applicant is a corporate body.
   See note (d)) — YES

0113389 10-Aug-04 08:29

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	0
Description	8
Claim (s)	0
Abstract	0
Drawing (s)	10 only

10. If you are also filing any of the following, state how many against each item.

Priority documents

Translations of priority documents

Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*

Request for preliminary examination and search *(Patents Form 9/77)*

Request for substantive examination *(Patents Form 10/77)*

Any other documents *(please specify)*

11. I/We request the grant of a patent on the basis of this application.

Signature Ursula Lenel Date 10-Aug-2004

12. Name and daytime telephone number of person to contact in the United Kingdom Ursula Lenel 01223-422290

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

Autofocus Methods

This is a continuation of a previous Patent Application GB0405773.3, filed 15th March 2004.

We now describe some specific control methods for autofocus cameras. For ease of explanation of operation of the control methods, we represent them schematically by way of flowcharts (Figs. 1 to 10). How the methods are implemented in an actual camera is a matter of engineering choice, and could be.for example, hard-wired control logic, a set of gates, registers and memory cells in a Field programmable logic array, or a general purpose DSP (digital signal processor) or microprocessor connected to input and output circuitry, executing a program that causes the ensemble to enact the control method(s), but other possibilities exist and are to be understood to be included in this invention.

Again for ease of description we represent each of the autofocus control methods as a sub-method of operation of the entire camera system, autofocus itself simply being one of the many operations necessary for a camera to perform a useful function, the other operations possibly including, calibration of the sensitivity to light and appropriate closure of the aperture and/or change of exposure time, the "taking" of the picture (i.e. the final exposure to be captured and stored), and the transfer of the captured image from the image sensor of the camera to some storage medium. For this reason the flowcharts describing the autofocus control methods have START and RETURN control points. The START point is where the overall or general camera control system begins the autofocus process, and the RETURN point is where this iteration of the autofocus process is complete and general camera control method (hereinafter the General Camera Control Method or GCCM) continues before later re-entering the autofocus control method (hereinafter the Autofocus Control Method or ACM) again.

Fig. 1 is a simplified generic ACM flowchart which underlies all of the more detailed methods described subsequently. To avoid repetitive description of operations, certain points in the control method are given labels of the form *Labxxx* where *xxx* is a string of one or more digits, and where the description says *goto Labxxx* it is to be understood that the operations at and after the label *Labxxx* are then to be carried out.

Once the ACM is entered via the START box, the first operation is to initialize to suitable values the loop variables that are used to control the details of the rest of the method. The ACM also has a *State* variable that persists from one invocation of the method to the next, although the ACM itself may change the value of the *State* variable. The essential point is that when the method returns control to the GCCM, the *State* is not lost, and unless deliberately changed by an external system e.g. the GCCM itself, the *State* variable will have the same value the next time the ACM is initiated.

> Next the *State* variable is checked and if the ACM is in *Idling* state (which means "do nothing") the ACM returns control directly to the GCCM.

> If instead the *State* is *Initialisation* state (which means, "prepare for a new focus

measurement process cycle of operations"), then certain focus control variables are initialised to suitable values (including the *frames* variable) after which:

[*Lab01*] the *frames* variable is incremented. A test is then carried out to check if all the required *frames* have been done and:

> if so, the *State* is moved to the next state; i.e. the *State* variable value is changed;

in either case the ACM then returns control to the GCCM and passes back a value *currpos* which describes the current focus-lens position.

If instead the *State* is *Running* state, then a Figure of Merit (FOM) is determined for the image being received by the camera with the focus-lens at its current position (*currpos*). This newly derived FOM is then compared with any previous FOM values determined since the last *Initialisation* state:

> if the new FOM is the best so far, then it is remembered as a new value of best FOM so far, for future FOM comparisons;

in either case, the ACM then works out how to alter the focal-lens position on the basis of this FOM measurement. Then *goto Lab01*.

If instead the *State* is *Flyback*, then the ACM causes the focus-lens to move to a known position. Then *goto Lab01*.

If instead the *State* is *Track Focus* then the *frames* variable is set to a large number, so that the rest of the algorithm will process many frames before it registers all required frames done. Then *goto Lab01*.

Otherwise, if in none of the above states, set *State* to *Idling* and RETURN to the GCCM.

The above generic description of the ACM does not describe in detail several crucial processes, which are now discussed in the more specific versions of the ACM illustrated by the flowcharts in Figs. 2 to 10.

Fig. 2 shows the first more specific implementation of the ACM described in general above. So in the first specific implementation a description of the loop variables and their initialised values can be seen. These include the number of initial steps, *initSteps* = 5 and flyback steps, *flybackSteps* = 5, the number of scan steps, *scansteps* = 25 , a zero value for the frames done variable *frames* and an upper limit for the drive value, *DrivMax* = 255, which is specific to each and every implementation, and simply represents the maximum allowable value for the lens position drive signal.

When the state on entry to the routine is initialisation, *State* = *init*, then specific values for the focus variables are defined as follows: best position *bestPos* = 0, current position *currpos* = 0, best Figure of Merit *bestFOM* = 0, and number of frames *frames* = *initSteps*.

If in the running state, *State* = *running*, then a current FOM value *thisFOM* is calculated by a call to a sub-process *getFOM* (described below in two preferred formats, but any suitable method of determining an FOM can be used here); this FOM is then checked against the current best FOM, *bestFOM*, and if the new value is better, then *bestFOM* is updated with the

value of *thisFOM*, and the focus lens position for best FOM, *bestPos* is updated from the current position *currPos*. Then a new lens position *currPos* to try is calculated, and the *frames* variable updated.

If the system is in the flyback state, *State = Flyback*, then the next lens position is set to 0, *currpos* = 0, and the *frames* variable set to a suitable value for the flyback process, =*flybacksteps* in this version, after which the *StateFrame* variable is incremented (NB in the flowcharts the notation ++x is used to indicate that the variable *x* is incremented (by 1)).

If the system is in the tracking state, *State = track*, then the *frames* variable is set to 100 in this version, before the *Stateframe* variable is incremented.

These latter states than pass to a test to see if all required frames are done by comparing *Stateframe* with *frames*, and if so, the system state is modified according to its current state (*init* changes to *running*, *running* to *flyback*, *flyback* to *Idle*, *Track* to *Track* (i.e. no change) and *Idle* to *Idle* (again, no change of state)). The last action is to move the lens to the newly computed position *currpos* by a call to the focal lens movement control routine *Focus()*.

If the entry state was any other state, then the state is set to *Idle* before returning.

Note that in the above description all the constant values to which the various variables are set to during the process are example values that have ben found to work well, but other values are also possible and useful in real implementations and these given here are merely guides to one particular implementation and are not meant to be limiting.

Fig. 3 shows a second variant of the basic autofocus algorithm, very similar to that in Fig. 2, the principal changes being:

In the initialization phase :-

initSteps = *flybacksteps* = 3 [instead of 5];

ScanSteps = 15 [instead of 25];

In the running state, the new position is calculated differently:-

*currpos = (StateFrame+1)*Drivemax/scansteps*;

Fig. 4 shows another slight variant of the method shown in Fig. 2, the principal differences being:

In the running state, the new position is calculated differently:-

*currpos = StateFrame*Drivemax/scansteps*;

After the test for all frames done (*Stateframe>frames?*)

an extra test is inserted if the state is *running*, to see if the best position is greater than a certain large value (in this example, *bestPos > 230?*), and if the test succeeds then instead of moving to flyback state instead the system moves to idle state, after resetting the next position to the latest best-position, *currpos = bestPos*, just prior to the instruction to move to that new position.

Alternatively, if the state is flyback, then state is reset to idle and again *currpos* is set to *bestPos*.

Fig. 5 is much like Figs. 2, 3, and 4, with combinations from each, with one additional

change:

After all frames are done and the state is flyback:

Instead of changing the state to idle, the state is left as flyback before returning control to the GCCM.

Note that each of these variants shown in Figs. 2, 3, 4, 5 each have their own slightly different behaviour which are advantageous in certain situations and are all useful methods in their own right.

Fig. 6 is yet another variant of autofocus control method, with an element of dead-reckoning built into it, designed to reduce the effects of any hysteresis in the actuator that converts an electrical drive signal into a mechanical position for the focal-lens. Clearly hysteresis in such a component will cause the lens to move to ***different*** positions for the same drive signal due to the previous history of mechanical and drive states, so that depending on these histories, methods ignoring hysteresis will behave differently at different times, and in general less well than if hysteresis were absent. The principal differences from the previous methods described are:

In the tracking state:-

the method uses the same control state sequence as the flyback state;
after the test for all frames done has been satisfied, if the system is in the running state, then the next new position of lens, *currpos* is computed by use of a look up table (*LUT*) using the stored *bestPos* value is the index into the table. This lookup table will contain pre-computed (static at design, assembly or test time, but possibly dynamically controlled values of position data, estimated to correct for the actual hysteresis of the actuator element).

Fig. 7 shows a simplified method of continuous ***tracking*** position control, which continually attempts to bring the image into focus despite changes in camera position and changes of scene in the view of the camera. Its operation is as follows:

Once started (*START*), a position step size *S* (which is the amount the lens drive signal will change during the next tracking iteration) is set to an initial value *S0*.

An initial "current" direction of step (lens in or lens out) is chosen.

[*StepPosition:]* Next the lens is stepped by *S* in the current direction.

At this point a focus Figure of Merit (focus quality) is measured, and if this FOM has ***not*** changed by a large amount (what is considered a large amount is predetermined and stored as a constant as part of the method control), then the step size *S* is ***increased*** by an amount *delta1*, but not allowed to exceed a maximum step size *Smax* whereupon control loops back to the *StepPosition* action.

If the FOM ***did*** change by a large amount and if the focus got worse (i.e. FOM reduced) then step size *S* is increased by an amount *delta2* again with a check to ensure it remains bounded by *Smax* before looping back to *StepPosition*

again.

If the focus ***improved*** after the last big change in FOM, then the stepsize *S* is ***decreased*** by an amount *delta3*, but constrained not to fall below a minimum value *Smin*. The step ***direction*** is then reversed, before looping back to *StepPosition* once more.

The key points to note about this method are:

a) There is no ***termination condition*** as such - the autofocus tracking process just continues *ad infinitum* until stopped by some external event - the subtle changes in step size give the process stability. This absence of termination condition prevents the method from falsely terminating on focus (FOM) plateaus or getting stuck in local minima, because inherent random noise in any real camera/imaging system will cause sufficient changes in the FOMs measured to allow the loop to escape from such conditions.

b) The step size can ***increase*** (up to some maximum *Smax)* ***while*** the method is operating. Even if poor focus decisions have been made in the past, due maybe to an object passing into the field of view transiently, or to camera movement e.g. shake, or to the object of interest moving in and out of focus, or even due to some sort of error in the whole system, the gain of the system (i.e. how much the focus position is shifted in response to a certain change in FOM) can still recover (i.e. increase) to bring the desired object back into focus in a reasonable time, and importantly to also help the system out of a transient false minima.

c) The ability to detect a ***large*** change in focus quality allows the method to detect a scene change (i.e. field of view change). When this occurs the step size is increased to allow a focus position search for a new object in view. Even if the change of scene is not detected outright, the continuous running of the method with no termination step enables a new object (position) to be brought into focus in a reasonable time.

Some further important aspects of this last mentioned focus tracking method are:

1) The largest allowable stepsize (*Smax*) should be constrained so that it is impossible to completely miss focussing on an object in the field of view by stepping over the lens position at which it is in focus, in one step. This maximum stepsize *Smax* can be calculated at system design time for any given sensor-chip+lens+actuator+drive electronics combination. A useful and preferable value of *Smax* has been found to be around $1/6^{th}$ of full-scale focal lens displacement.

2) The minimum stepsize *Smin* is more difficult to pre-estimate at system design time. However, it is preferably smaller than half full-scale focal lens displacement. A useful and preferable value of *Smin* has been found to be around $1/32^{nd}$ of full-scale focal lens displacement.

3) For an autofocus control system where the focal position drive signal is in the range 0 to 255 (e.g. a digital system with a one-byte control word), the following values for each of the parameters of this control method are preferable (and for other overall control ranges similar preferable values may be scaled from these):

initial stepsize	*S0*	= 48
minimum stepsize	*Smin*	= 8
maximum stepsize	*Smax*	= 48
first stepsize increment	*delta1*	= 16
second stepsize increment	*delta2*	= 6
third stepsize increment	*delta3*	= 8

Fig. 8 shows in more detail a practical implementation of a tracking focus method, which lacks some of the sophisticated details of the general method just outlined, but which is nonetheless useful in its own right. As shown here the autofocus process method appears to have simple termination points (the round-cornered terminal boxes at middle right and bottom left of Fig. 8 containing the word "return"), but in practice this is simply so that the external controlling process (the GCCM e.g.) may easily choose to terminate the otherwise looping process when required. So in practice the GCCM would call this process (shown in Fig. 8) repeatedly (as if the Fig. 8 process itself continuously looped between the entry box (round cornered box at top left containing "AF_Track Slow()) and the previously mentioned exit terminal boxes), with some GCCM determined parameter causing termination, not the autofocusing process itself.

As in previous methods described, the system has a *State* parameter which largely determines the gross effect of an activation of the method, but a particular difference, due to the need during tracking of continuous looping through the method, is the very first condition test *(first call?)*, which ensures that two parameters, *currpos* and *Stepsize* are initialised only on the very first activation of the method, subsequent activations retaining the previous state or value of these. This method incorporates directly one or other of the previously described autofocus methods (as described with respect to Figs. 2, 3, 4, 5 or 6) which occurs after the conditional test *(a/f fn defined?)=>YES* in the process box containing *(call the a/f fn [e.g. Algo1/2/3, NoComp, deadreck])*, where the correspondence is between *Algo1* and Fig. 2, *Algo2* and Fig. 3, *Algo3* and Fig. 4, *NoComp* and Fig. 5, or *deadreck* and Fig. 6. The conditional test *(a/f fn defined?)* simply checks for the existence of a definition of which one of these functions to use, and if it doesn't exist, calls none of them.

A new feature in the Fig. 8 method not seen in earlier versions described, is an FOM history memory, in this example implemented as an array of four FOM values *FOMhist[0:3]*, together with an index *histptr* into this array which determines where the next value will be stored. During operation, to reduce the effects of noise and camera/scene movement, a number of (four in the example shown) successive FOMs are recorded and stored in the *FOMhist[]* array, before any changes to focal position via *currpos* are made. This makes the method somewhat slower than otherwise, but the improvement in reliability and stability is worthwhile in many situations. The choice of number of such averaging iterations may of course be varied to suit any particular application. When, in this example, four FOMs are in the array, an average *curravg* over the four values is computed, *histptr* is reset to zero, and a check for reaching maximum lens position (hitting "endstops") is carried out. The currently

computed average FOM is compared with the previous average (if it exists, and if it doesn't then one is generated from the current value, stored as *lastavg* and used for the next iteration) and depending on whether the latest average FOM is better (more in focus) or worse than the last, and whether the lens was last moved in the positive or negative direction, the new position to be used, *currpos*, is produced either by incrementing or decrementing the current value by *Stepsize*, with the resulting value clipped to the allowable range (in this example between 0 and 255). The logic is clear from the flow diagram at Fig. 8. The method finishes by moving the lens to the new position *currpos*, before starting the next iteration if so allowed by the GCCM.

Fig. 9 shows a more detailed implementation incorporating some of the features of the adaptive tracking process with *dynamic* changes to the stepsize as outlined above with respect to Fig. 7, and incorporating several of the features of the non-adaptive tracking method outlined with respect to **Fig.** 8, including the FOM history store and FOM averaging, and the first-call-only initialisation of certain parameters, and most particularly the parameters *currpos, currstep* (which replaces *StepSize* in this method), and *stepdec*.

It will be seen that in several places in the method the size of step *currstep* to be next made in focal lens position, is altered as a function of the state of other parameters (NB the notation in several of the process boxes like those near bottom right of Fig. 9, *viz.* (*currpos += currstep*) and (*currpos -=currstep*) is standard C-language notation meaning that the variable *currpos* is to be incremented (+=) or decremented (-=) by the amount *currstep*). The value of *currstep* is also kept within bounds by the various checks against *maxstep* and *minstep*.

As with the previously described autofocus methods, this method illustrated in Fig. 9 enlists a sub-process *getFOM()* to determine a value for the Figure of Merit at the current position. Examples of such FOM determining methods will now be described.

Fig. 10 shows a method (called here AF_LOGFOM1() but which may be substituted for the generic sub-method *getFOM()* enlisted by the herein previously described autofocus methods) for determining a Figure of Merit. In this example the method works on an 8x8 (total block size = 64) block of image pixels which have been Discrete Cosine Transform (DCT) encoded (e.g. as may be found in common compressed image data formats). The two numeric blocks at top right of Fig. 10 represent the layout of a typical 8x8 block of elements in the complete set of DCT elements, with the first numeric block showing the index scheme used within the 8x8 block (indexes running sequentially from 0 through 62, and the second numeric block showing the coefficients used at each index position, in the method. Note that in each block certain cells (15 in each) are greyed: the top set of greyed cells (i.e. in the top numeric block) are the indexes of those cells, the bottom set of greyed cells (i.e. in the lower numeric block) are the only non-zero coefficients used, keyed to the corresponding indexes in the top numeric block. Cells where the coefficients are zero are ignored and play no part in the method.

In this example the total size of the image DCT values used is 176 x 144 pixels corresponding to a set of 22 x 18 = 396 blocks of 8x8=64 pixels each. These 396 blocks are processed in turn by the method. The list of 15 non-zero coefficients is loaded into array *coefflist[]*, and the list of 15 indexes of the locations of the non-zero coefficients is loaded into *indexlist[]*. The FOM output value *FOMout* is initialised to zero as is the block index *blkindx*, and the

partial FOM value ***FOMpart***, the latter within the outer loop of the method. On each trip round the outer loop the block index ***blkindx*** is incremented by the block size ***blksiz***, and the output FOM ***FOMout*** incremented by half the value of the partial FOM ***FOMpart***. The non-zero coefficient list index ***step*** is also reset to zero at each pass round the outer loop.

Within the inner loop each of the 15 non-zero coefficients ***coefflist[0:14]*** is multiplied by its corresponding DCT value ***DCTbuff[blkindx+indexlist[step]]*** to give product ***tempres***, where ***blkindx*** is an offset to the start of each 8x8 block of DCT values, and ***indexlist[step]*** is the corresponding offset within that block of the DCT value to be multiplied by the coefficient. The partial FOM value ***FOMpart*** is then incremented by the absolute value of ***tempres***.

There is also in the outer loop a block counter and when it determines that all blocks within the DCT buffer ***DCTbuff[]*** have been processed, the method returns the FOM estimate ***FOMout***.

Other much simpler methods of determining an FOM may also be used as sub-method ***getFOM()*** in the above autofocus methods. One-dimensional (1D) filters with as few as 9 taps work well, looking only at one-line of raw pixel data (i.e. not DCT values as used above) at a time. Alternatively two-dimensional filters looking at several lines of raw pixel data simultaneously will generally give improved performance over the 1D methods, as instances of ***getFOM()***, but the improvements are not always ***necessary***, quite reliable results being achievable with 1D methods.

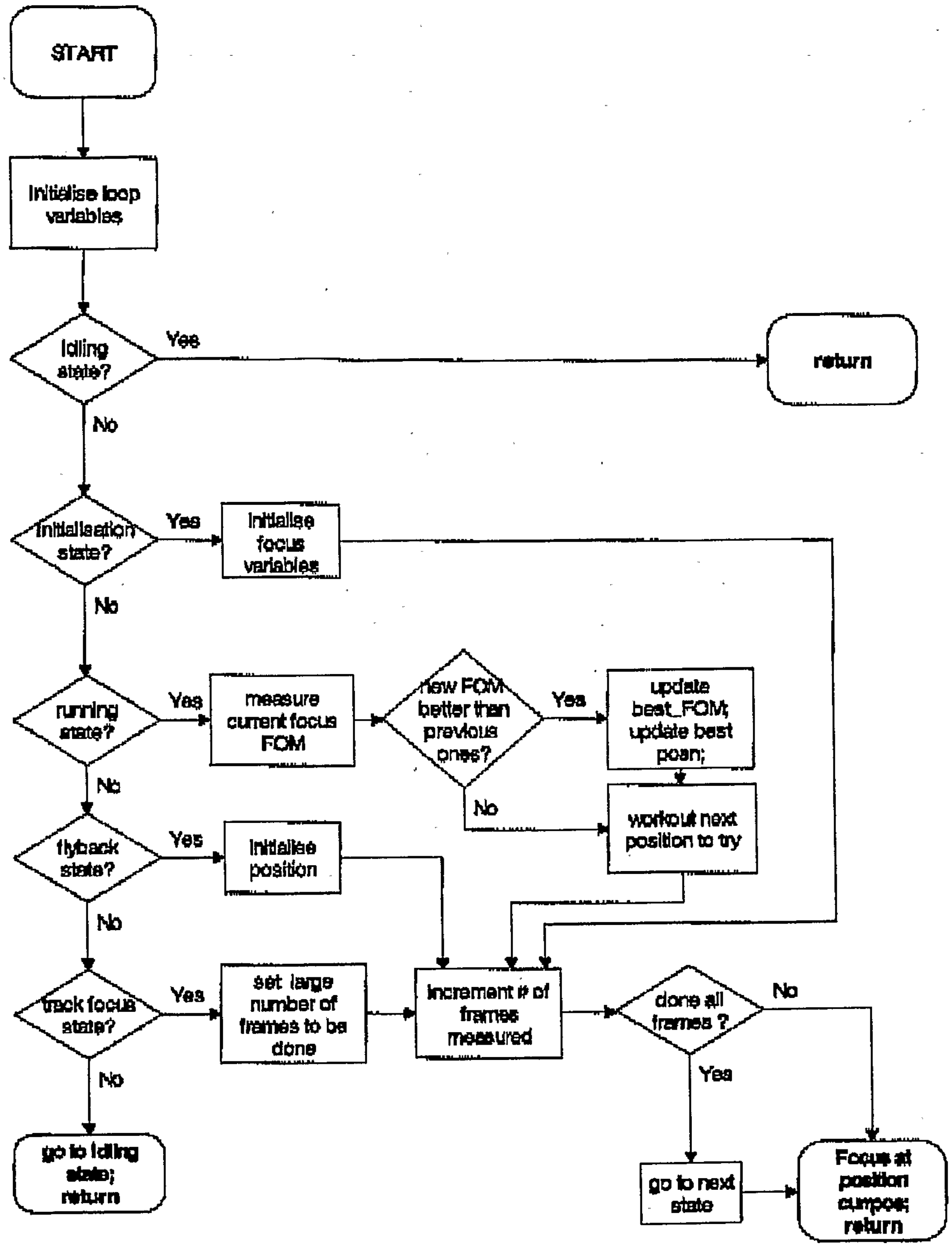

START
Initialise loop variables
Idling state?
Yes
return
No
Initialisation state?
Yes
Initialise focus variables
No
running state?
Yes
measure current focus FOM
new FOM better than previous ones?
Yes
update best_FOM; update best posn;
No
workout next position to try
No
flyback state?
Yes
Initialise position
No
track focus state?
Yes
set large number of frames to be done
Increment # of frames measured
done all frames ?
No
Yes
No
go to idling state; return
go to next state
Focus at position curnpos; return

Fig. 1

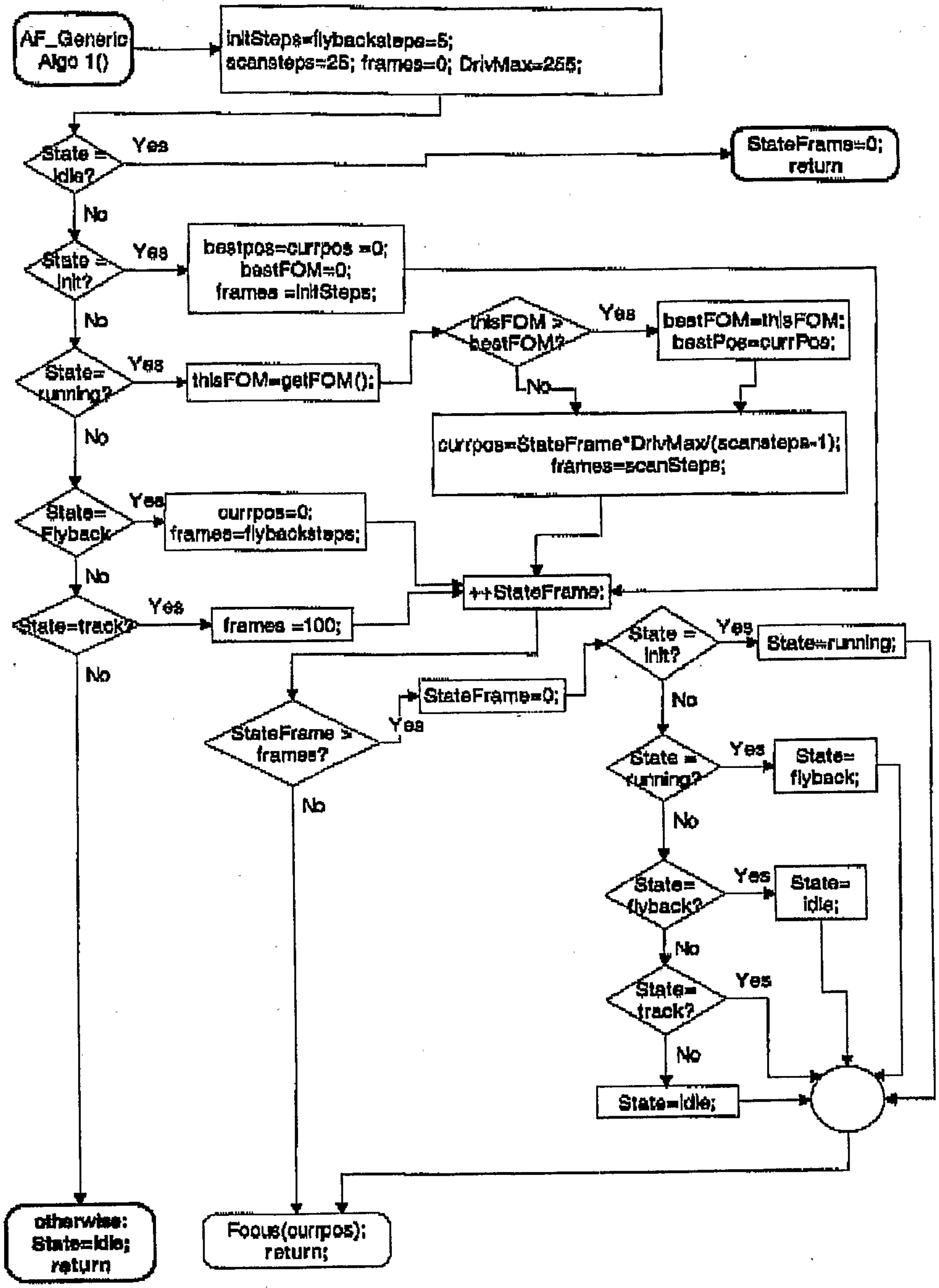
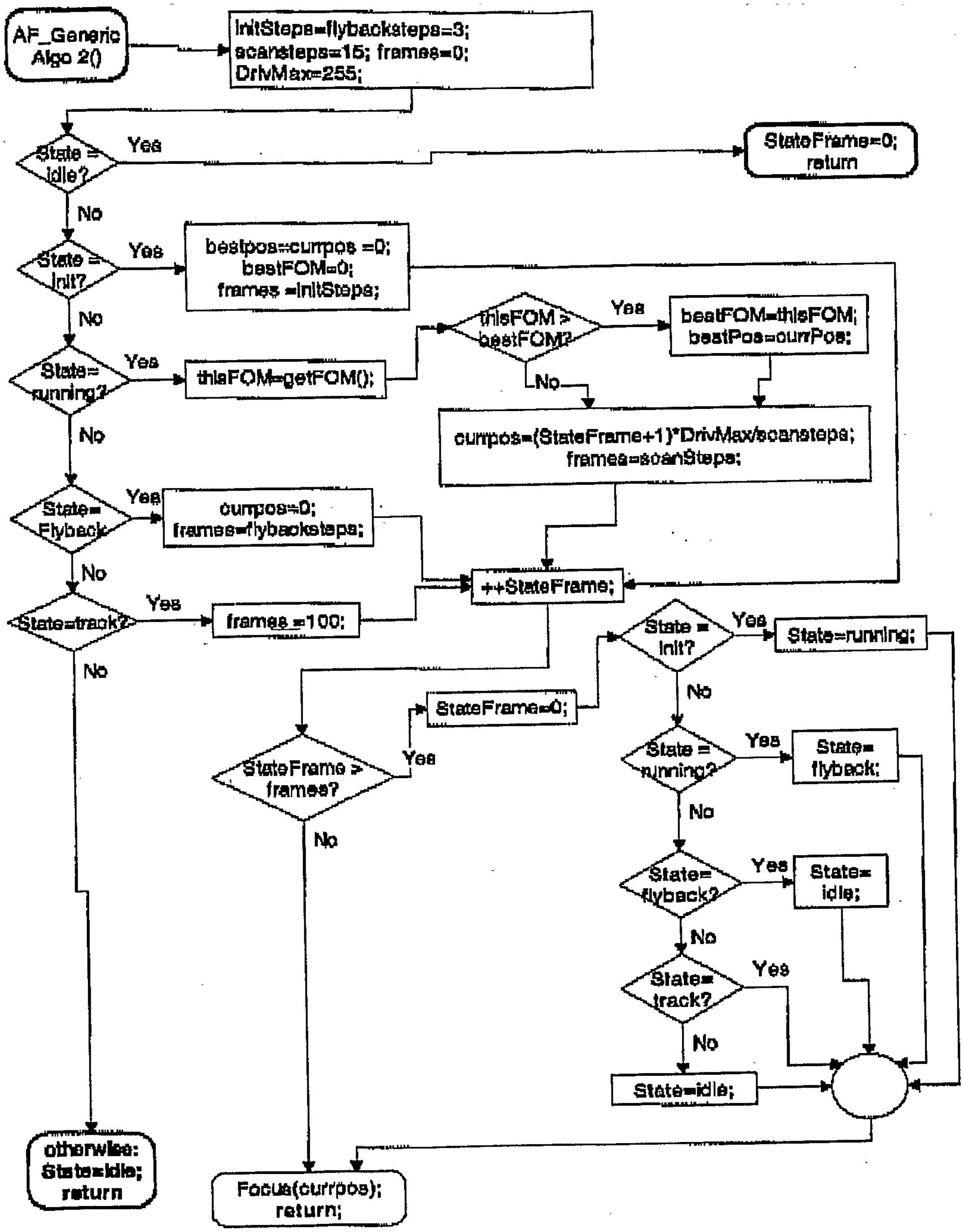

AF_Generic Algo 1()
initSteps=flybacksteps=5;
scansteps=25; frames=0; DrivMax=255;
State = Idle?
Yes
StateFrame=0; return
No
State = Init?
Yes
bestpos=currpos =0;
bestFOM=0;
frames =initSteps;
No
State= running?
Yes
thisFOM=getFOM();
thisFOM > bestFOM?
Yes
bestFOM=thisFOM;
bestPos=currPos;
No
currpos=StateFrame*DrivMax/(scansteps-1);
frames=scanSteps;
No
State= Flyback
Yes
currpos=0;
frames=flybacksteps;
No
State=track?
Yes
frames =100;
++StateFrame;
No
StateFrame > frames?
Yes
StateFrame=0;
State = Init?
Yes
State=running;
No
State = running?
Yes
State= flyback;
No
State= flyback?
Yes
State= idle;
No
State= track?
Yes
No
State=idle;
No
otherwise:
State=idle;
return
Focus(currpos);
return;

Fig. 2



AF_Generic Algo 2()
InitSteps=flybacksteps=3; scansteps=15; frames=0; DrivMax=255;
State = idle?
Yes
StateFrame=0; return
No
State = Init?
bestpos=currpos =0; bestFOM=0; frames =InitSteps;
State= running?
thisFOM=getFOM();
thisFOM > bestFOM?
bestFOM=thisFOM; bestPos=currPos;
currpos=(StateFrame+1)*DrivMax/scansteps; frames=scanSteps;
State= Flyback
currpos=0; frames=flybacksteps;
State=track?
frames =100;
++StateFrame;
StateFrame > frames?
StateFrame=0;
State = Init?
State=running;
State = running?
State= flyback;
State= flyback?
State= idle;
State= track?
State=idle;
otherwise: State=idle; return
Focus(currpos); return;

Fig. 3

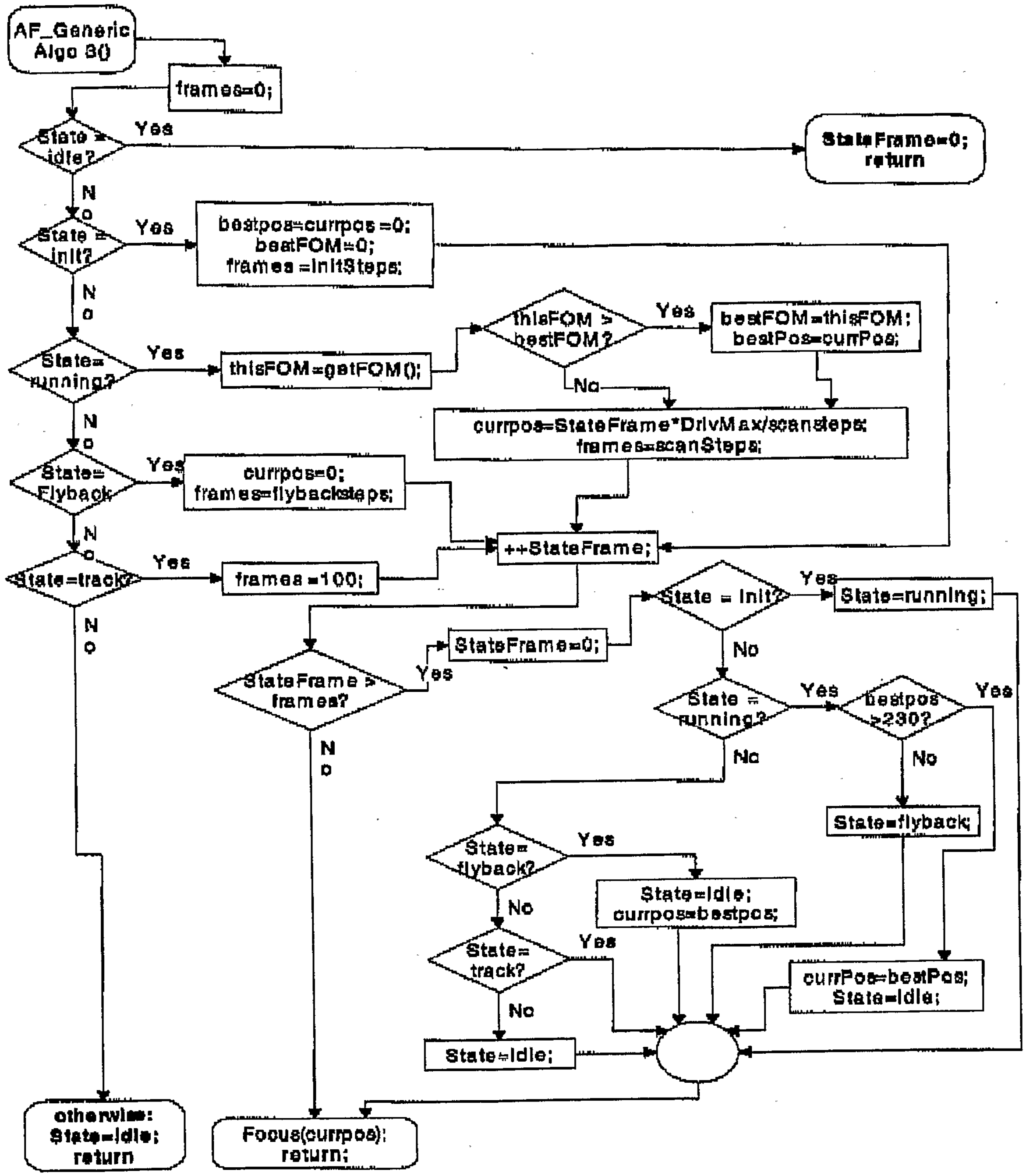

AF_Generic Algo 3()
frames=0;
State = idle?
Yes
StateFrame=0; return
No
State = init?
Yes
bestpos=currpos=0; bestFOM=0; frames=InitSteps;
No
State= running?
Yes
thisFOM=getFOM();
thisFOM > bestFOM?
Yes
bestFOM=thisFOM; bestPos=currPos;
No
currpos=StateFrame*DrivMax/scansteps; frames=scanSteps;
No
State= Flyback
Yes
currpos=0; frames=flybacksteps;
No
State=track?
Yes
frames=100;
++StateFrame;
No
StateFrame > frames?
Yes
StateFrame=0;
State = Init?
Yes
State=running;
No
State = running?
Yes
bestpos >230?
Yes
No
State=flyback;
No
State= flyback?
Yes
State=Idle; currpos=bestpos;
No
State= track?
Yes
No
State=Idle;
currPos=bestPos; State=Idle;
No
otherwise: State=Idle; return
Focus(currpos); return;

Fig. 4

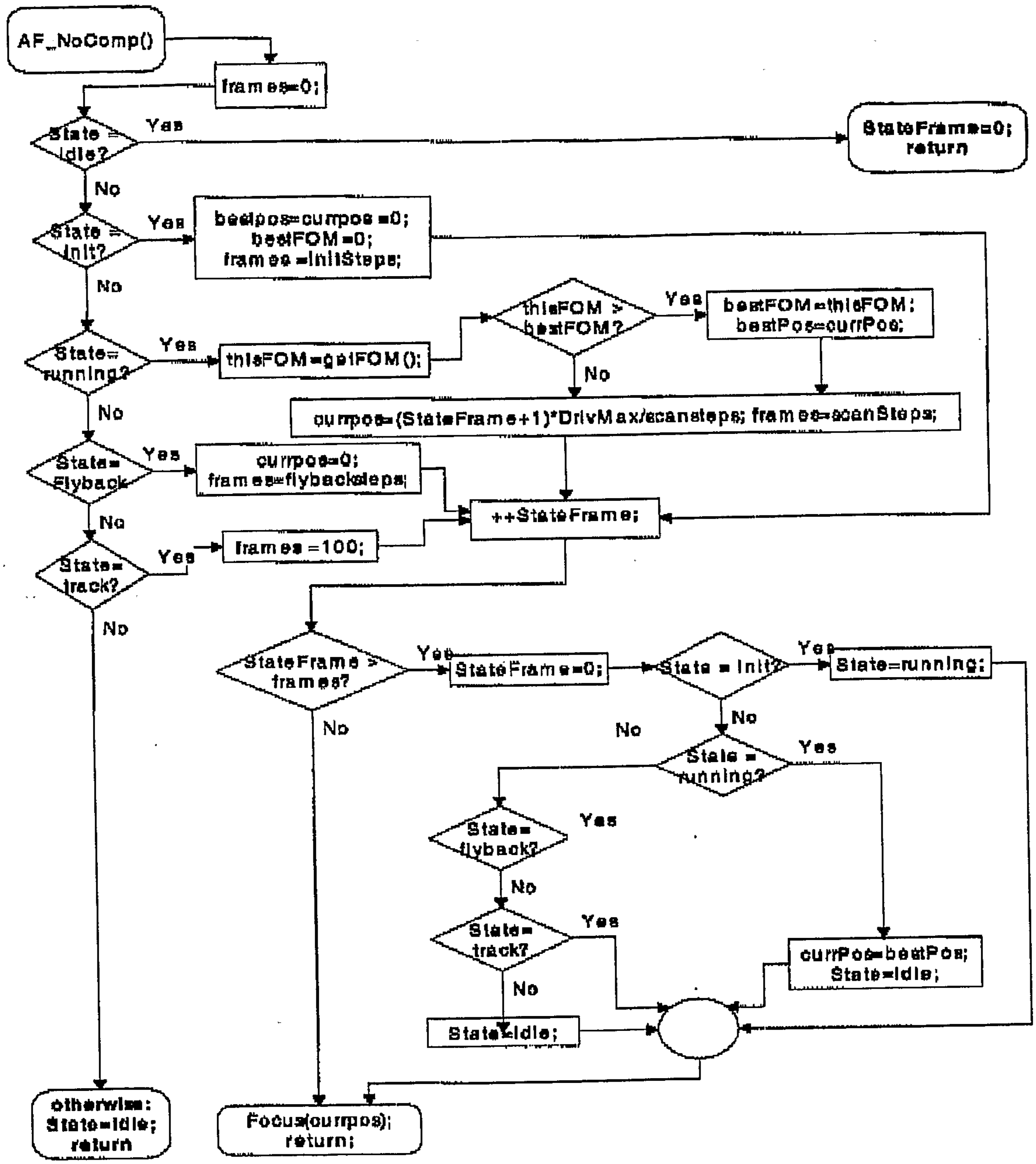

AF_NoComp()
frames=0;
State = Idle?
Yes
StateFrame=0;
return
No
State = Init?
Yes
bestpos=currpos =0;
bestFOM =0;
frames =initSteps;
No
State= running?
Yes
thisFOM=getFOM();
thisFOM > bestFOM?
Yes
bestFOM=thisFOM;
bestPos=currPos;
No
currpos=(StateFrame+1)*DrivMax/scansteps; frames=scanSteps;
No
State= Flyback
Yes
currpos=0;
frames=flybacksteps;
++StateFrame;
No
State= track?
Yes
frames =100;
No
StateFrame > frames?
Yes
StateFrame=0;
State = Init?
Yes
State=running;
No
No
State = running?
Yes
State= flyback?
Yes
No
State= track?
Yes
No
State=Idle;
currPos=bestPos;
State=Idle;
otherwise:
State=Idle;
return
Focus(currpos);
return;

Fig. 5.

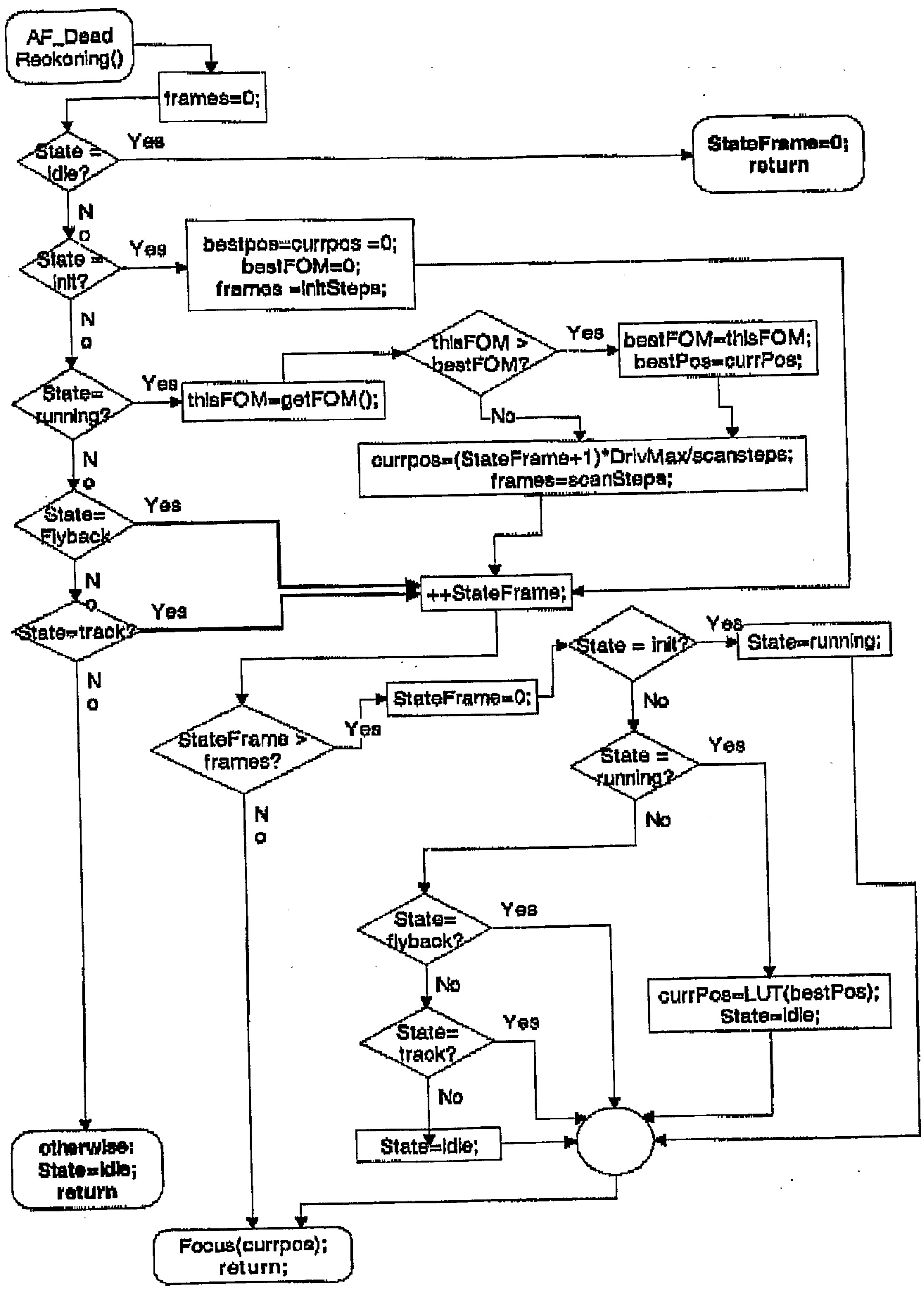

AF_Dead Reckoning()
frames=0;
State = Idle?
Yes
StateFrame=0; return
No
State = init?
Yes
bestpos=currpos =0; bestFOM=0; frames =initSteps;
No
State= running?
Yes
thisFOM=getFOM();
thisFOM > bestFOM?
Yes
bestFOM=thisFOM; bestPos=currPos;
No
currpos=(StateFrame+1)*DrivMax/scansteps; frames=scanSteps;
State= Flyback
Yes
No
State=track?
Yes
++StateFrame;
State = init?
Yes
State=running;
No
StateFrame=0;
StateFrame > frames?
Yes
State = running?
Yes
No
No
State= flyback?
Yes
No
currPos=LUT(bestPos); State=Idle;
State= track?
Yes
No
State=Idle;
otherwise: State=Idle; return
Focus(currpos); return;

Fig. 6

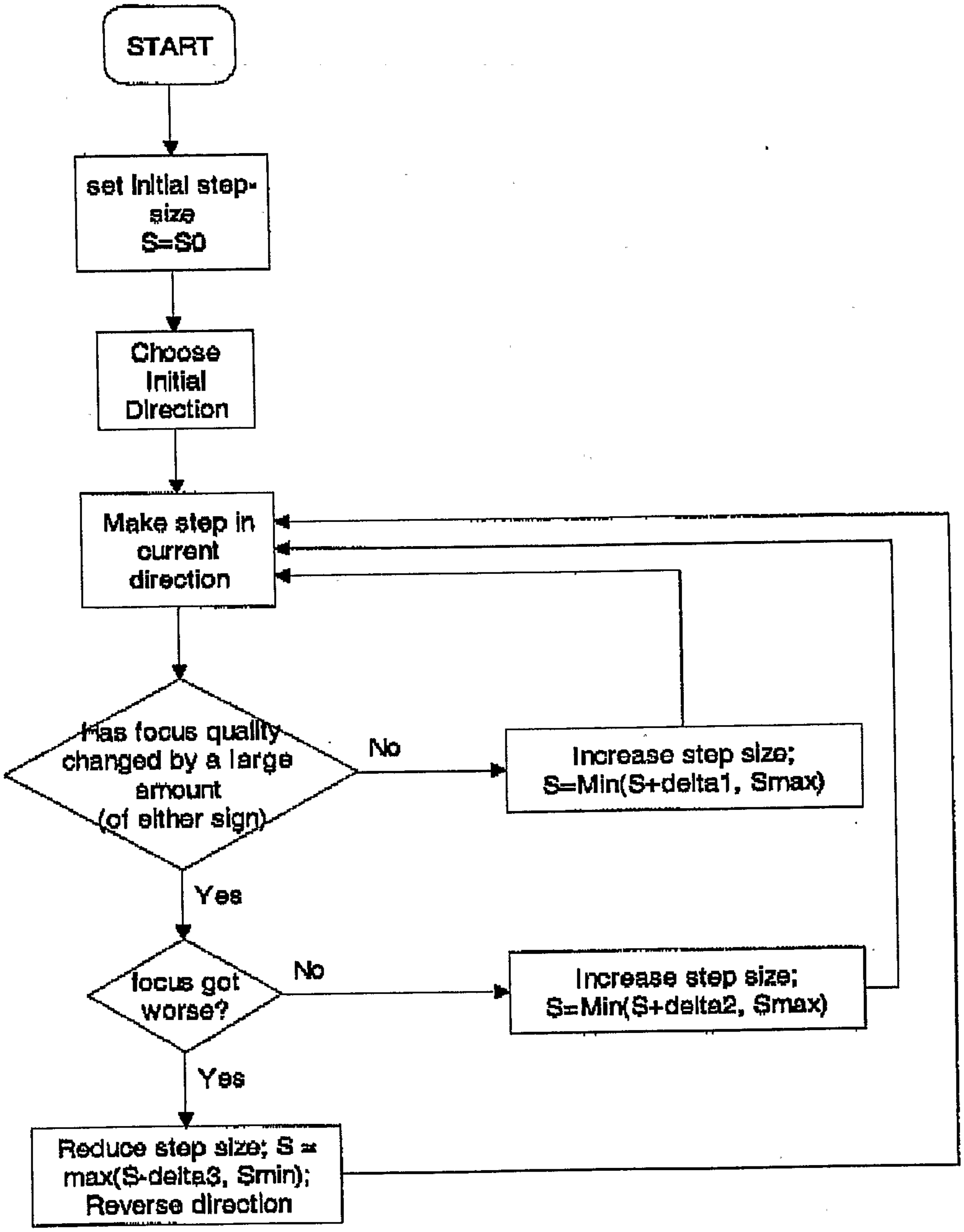

START
set initial step-size S=S0
Choose Initial Direction
Make step in current direction
Has focus quality changed by a large amount (of either sign)
No
Increase step size; S=Min(S+delta1, Smax)
Yes
focus got worse?
No
Increase step size; S=Min(S+delta2, Smax)
Yes
Reduce step size; S = max(S-delta3, Smin); Reverse direction

Fig. 7

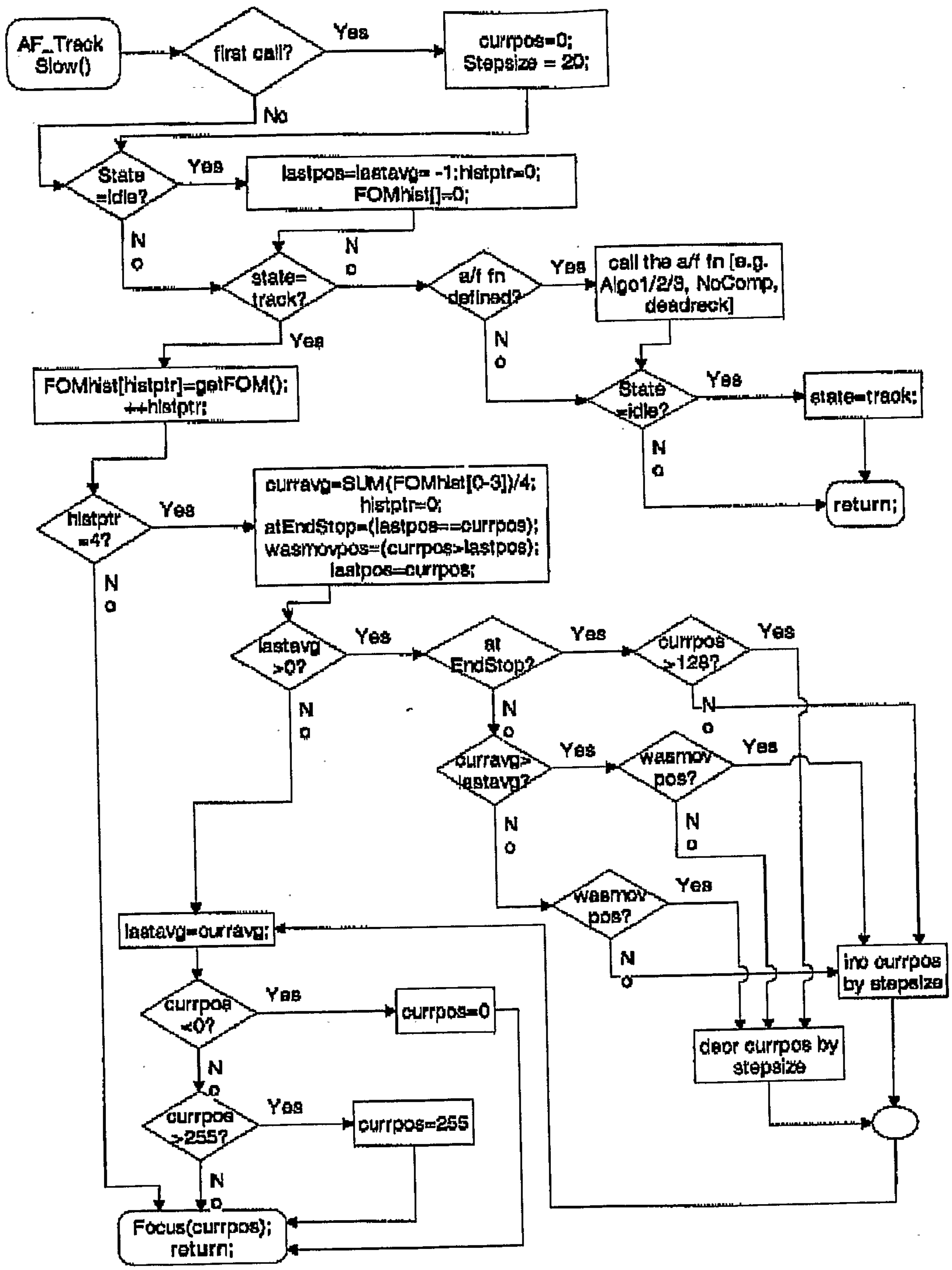

AF_Track Slow()
first call?
Yes
No
currpos=0; Stepsize = 20;
State =idle?
Yes
lastpos=lastavg= -1;histptr=0; FOMhist[]=0;
No
state= track?
No
a/f fn defined?
Yes
call the a/f fn [e.g. Algo1/2/3, NoComp, deadreck]
Yes
FOMhist[histptr]=getFOM(); ++histptr;
No
State =idle?
Yes
state=track;
No
return;
histptr =4?
Yes
curravg=SUM(FOMhist[0-3])/4; histptr=0; atEndStop=(lastpos==currpos); wasmovpos=(currpos>lastpos); lastpos=currpos;
No
lastavg >0?
Yes
at EndStop?
Yes
currpos >128?
Yes
No
No
No
curravg> lastavg?
Yes
wasmov pos?
Yes
No
No
wasmov pos?
Yes
No
lastavg=curravg;
inc currpos by stepsize
decr currpos by stepsize
currpos <0?
Yes
currpos=0
No
currpos >255?
Yes
currpos=255
No
Focus(currpos); return;

Fig. 8

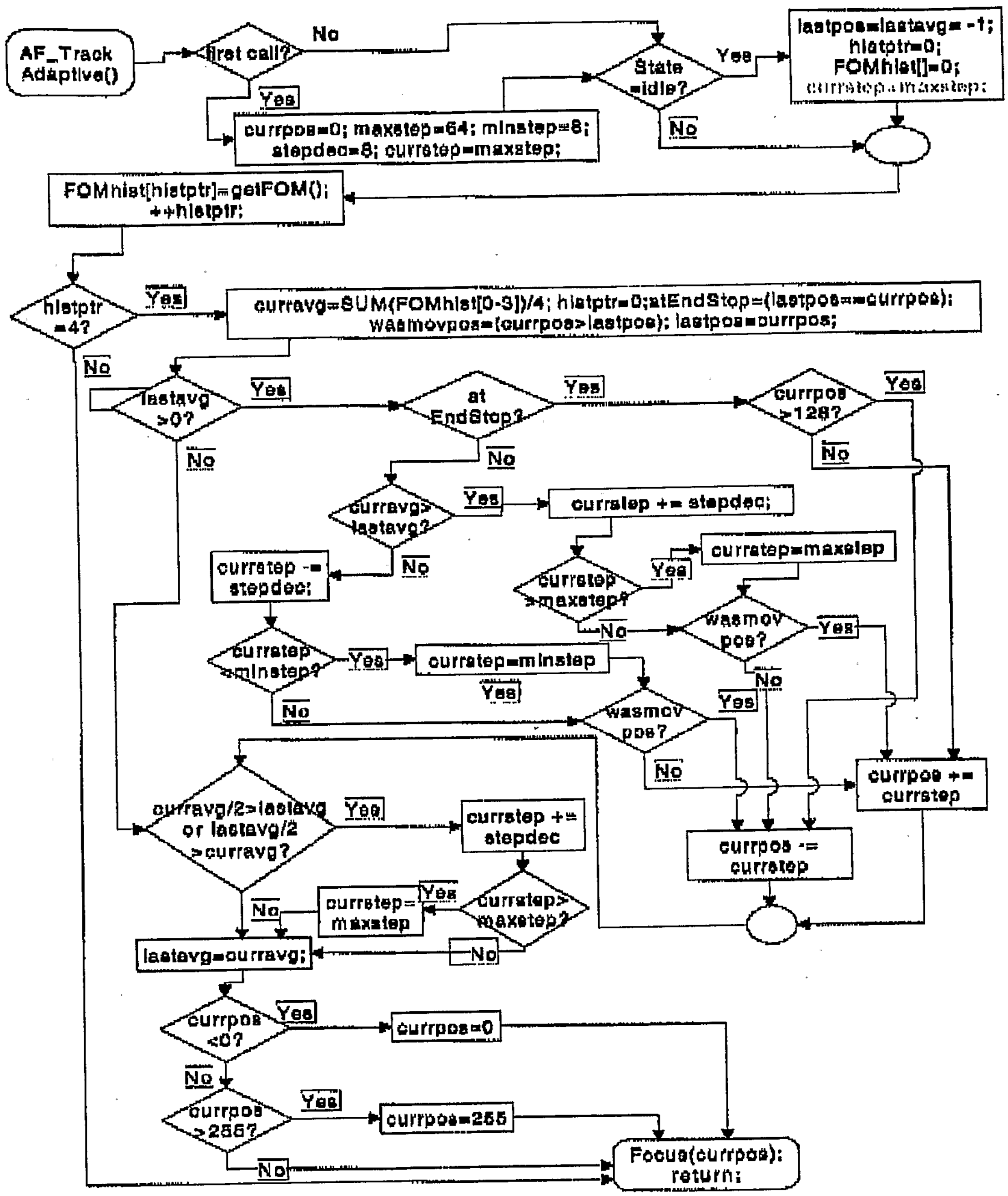

AF_Track Adaptive()
first call?
No
Yes
currpos=0; maxstep=64; minstep=8; stepdec=8; currstep=maxstep;
State =idle?
lastpos=lastavg= -1; histptr=0; FOMhist[]=0; currstep=maxstep;
FOMhist[histptr]=getFOM(); ++histptr;
histptr =4?
curravg=SUM(FOMhist[0-3])/4; histptr=0;atEndStop=(lastpos==currpos); wasmovpos=(currpos>lastpos); lastpos=currpos;
lastavg >0?
at EndStop?
currpos >128?
curravg> lastavg?
currstep += stepdec;
currstep -= stepdec;
currstep >maxstep?
currstep=maxstep
wasmov pos?
currstep <minstep?
currstep=minstep
currpos += currstep
currpos -= currstep
curravg/2>lastavg or lastavg/2 >curravg?
currstep += stepdec
currstep> maxstep?
currstep= maxstep
lastavg=curravg;
currpos <0?
currpos=0
currpos >255?
currpos=255
Focus(currpos); return;

Fig. 9

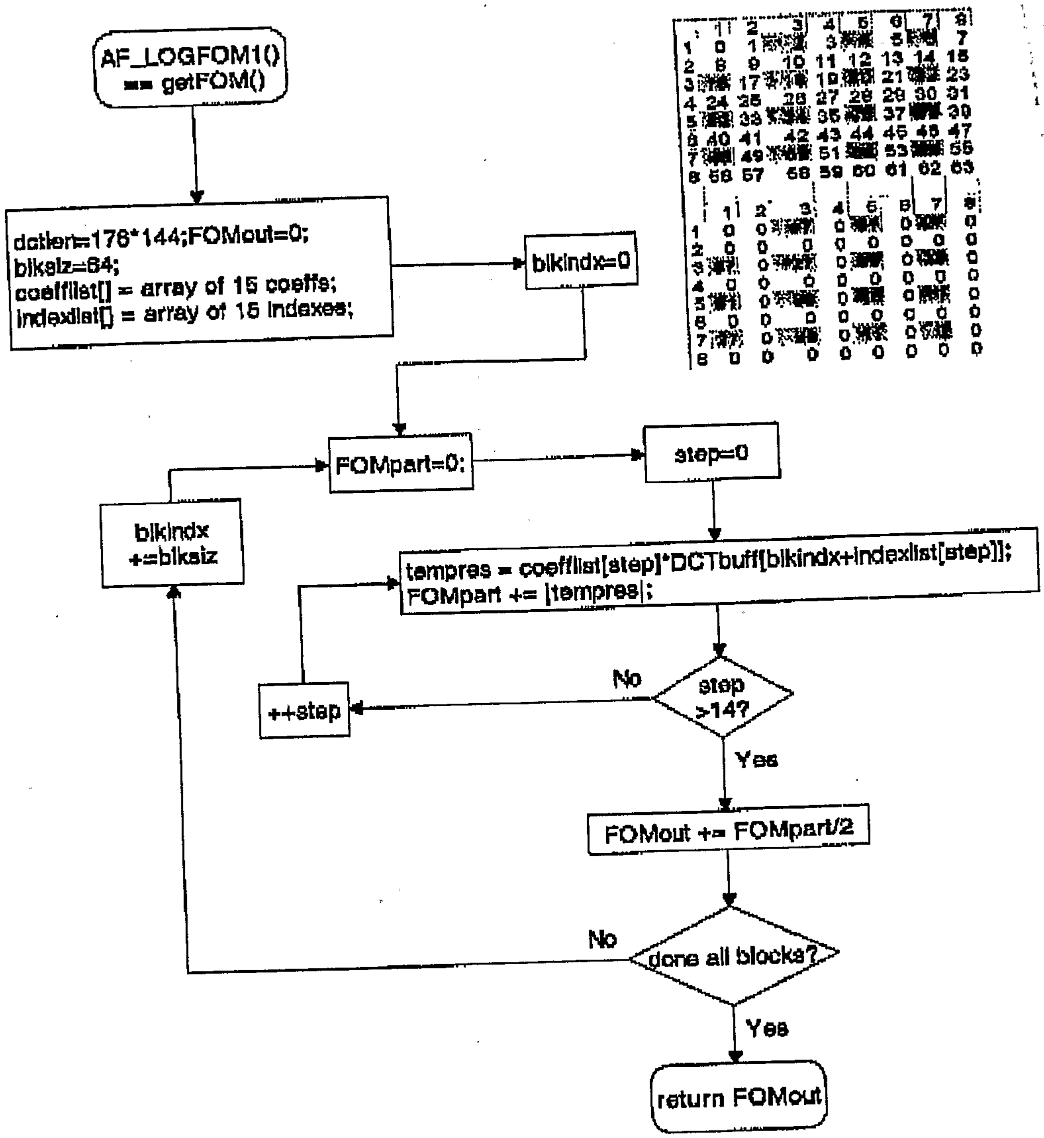

AF_LOGFOM1() == getFOM()
dctlen=176*144;FOMout=0;
blksiz=64;
coefflist[] = array of 15 coeffs;
indexlist[] = array of 15 indexes;
blkindx=0
FOMpart=0;
step=0
blkindx +=blksiz
tempres = coefflist[step]*DCTbuff[blkindx+indexlist[step]];
FOMpart += |tempres|;
No
step >14?
++step
Yes
FOMout += FOMpart/2
No
done all blocks?
Yes
return FOMout

Fig. 10